AD-A223 900

AN ARTIFACT FILTER FOR EVENT-RELATED POTENTIALS

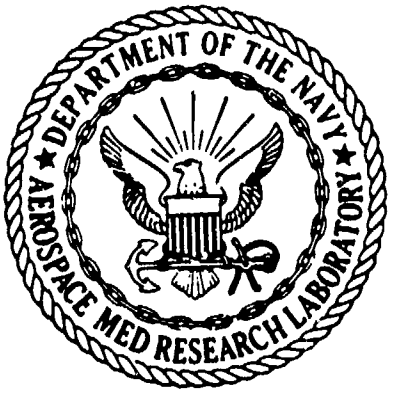
R.R. Stanny and S.J. LaCour

Naval Aerospace Medical Research Laboratory
Naval Air Station
Pensacola, Florida 32508-5700

Reviewed and approved 22 March 1990

J. A. Brady

J. A. BRADY, CAPT, MSC USN
Commanding Officer



This research was sponsored by the Naval Medical Research and Development Command under work unit 62233N MM33P30.001-7001.

The views expressed in this article are those of the authors and do not reflect the official policy or position of the Department of the Navy, Department of Defense, nor the U.S. Government.

Trade names of materials and/or products of commercial or nongovernment organizations are cited as needed for precision. These citations do not constitute official endorsement or approval of the use of such commercial materials and/or products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	April 1990	

4. TITLE AND SUBTITLE

An Artifact Filter for Event-related Potentials

5. FUNDING NUMBERS

62234N
RS34H21. 01

6. AUTHOR(S)

R. R. Stanny and S. J. LaCour*

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Aerospace Medical Research Laboratory
Naval Air Station, Bldg. 1953
Pensacola, FL 32508-5700

8. PERFORMING ORGANIZATION REPORT NUMBER

NAMRL Special Report 90-2

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Office of Naval Technology
800 N. Quincy Street
Arlington, VA 22217
Naval Personnel Research and Development Center
San Diego, CA 92152-6800

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

Navy Regional Data Automation Center, Pensacola, FL 32508-5700

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release;
distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Scalp-surface recordings of event-related potentials (ERPs) are frequently contaminated by electrical artifacts. We describe a Fortran 77 computer program that examines ERP data for several types of electrical artifacts: eyeblinks, voltages spikes, large local voltages, large overall voltages, amplifier saturation effects, and dead-amplifier effects. Where possible, the program compensates the ERP data for electrooculogram artifacts by time-domain cross-regression procedures.

=> Artifi, Biomedical Instrumentation

=> EEG, EOG, eyeblinks, Voltage spikes, Absolute Voltages, Root-mean-Square voltages, rms, amplifier clipping, Saturation

14. SUBJECT TERMS	15. NUMBER OF PAGES
Performance assessment, event-related potentials, evoked potentials, psychophysiology, ERP, Artifact Detection	40
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	

SUMMARY PAGE

THE PROBLEM

We are using event-related potentials (ERPs) to characterize the changes in cognition that occur when a high level of performance must be sustained for many hours despite fatigue and loss of sleep. The need for such performance occurs during the repeated, long-range tactical missions that characterize sustained and continous operations in naval aviation.

FINDINGS

We describe a computer program we have developed that scans ERP data for several types of electrical artifacts. When feasible, the program attempts to correct the electroencephalogram (EEG) for the effects of eye movements and blinks. After doing so, the program checks the success of its corrections, writes a data file of the remaining artifacts, and writes the corrected EEG to a data file. A copy of the program's source code is included as an appendix.

RECOMMENDATIONS

Clearly, ERP data must be routinely screened for electrical artifacts. The algorithms used in this program represent several contemporary approaches to the artifact-detection problem. None of these algorithms is fully satisfactory when used in isolation. Combining them, however, yields an improved quality-control system for ERP studies.

INTRODUCTION

An ERP is an aggregate electrical field produced when a population of neurons discharge simultaneously in response to a sensory or internal event. The amplitudes of ERPs are greatly attenuated as they propagate from the brain to the surface of the scalp. As a result, ERPs recorded at the scalp are easily contaminated by stray electrical events. These electrical artifacts can render data sets unusable. If they pass undetected, they can render a data set misleading. Consequently, artifact detection is among the most important aspects of ERP recording technique. For these reasons, we have written a computer program that scans ERP data for several types of electrical artifacts. This report describes our program.

We review the topic of electroencephalogram (EEG) and ERP artifacts only briefly; a comprehensive discussion of the topic can be found elsewhere (1). We use the term artifact here to refer to any potential that makes a sample of ERP data unusable, including random noise.

The amplitudes of ERPs recorded with scalp-surface electrodes are small, within roughly an order of magnitude of 1 uV. Hence, they are easily contaminated by extraneous voltages. Such artifacts may be biological potentials of nonneuronal origin or nonbiological potentials generated by the recording system or nearby electrical equipment. For example, artifacts in ERP recordings include potentials from eye movements and blinks, muscle contractions, tongue movements (the tongue is polarized end-to-end), galvanic skin responses, changes in the skin-electrode interface, and movements of electrode leads. Line-power artifacts occur when equipment is improperly grounded, electrode impedances are high, or electrode leads are placed too near power cords or other radiant sources. Aliasing artifacts occur when the EEG is converted to a digital format without first removing frequency components above one-half of the sampling rate (1).

Adjusting instrumentation and/or procedures will minimize or eliminate many artifacts. Others are difficult or impossible to avoid. Artifacts from subjects are usually least controllable: Subjects blink, move, and emit galvanic skin responses despite instructions to the contrary. Usually, the best one can do is to identify any artifacts and ensure that they do not compromise the data. Our computer program should assist in that effort.

This report contains three major sections and an appendix. Section 1 discusses artifact detection and correction techniques of the artifact filtering program, Artfil; section 2 describes how the program is used; and section 3 contains system requirements. The appendix includes a copy of the program's source code.

1. ARTIFACT DETECTION AND CORRECTION

ARTIFACT DETECTION

Artfil checks the EEG and electrooculogram (EOG) for six types of artifacts: (1) eyeblinks, (2) voltage spikes, (3) large absolute voltages, (4) large root-mean-square (rms) voltages, (5) dead EEG channels, and (6) amplifier clipping (or saturation).

1. Eye blink detection is performed using an algorithm described by Gratton, Coles, and Donchin (2). A blink is identified when the local slope of the vertical EOG trace exceeds a critical value, blinkcrit. A local EOG slope exceeding blinkcrit indicates eyelid movement and, hence, a blink. After Gratton et al., we define blinkcrit as a criterion change in voltage during a 10-ms interval.

2. Spike detection is performed similarly to blink detection. Spikes are identified when the difference between successive voltage-values in an EEG or EOG time series exceeds a criterion value, spikecrit. The vertical EOG is not scanned for spikes, which could be confused with blinks. If an EEG channel or the lateral EOG channel contains a spike artifact, the data are not corrected for ocular artifacts and are flagged for rejection.

3. Large absolute voltages are detected by comparing each voltage in the EEG time series to a criterion voltage, artcrit, after the eye-movement compensation algorithm described presently has been applied to the EEG data. Any epoch and channel of EEG data that contains an absolute voltage exceeding artcrit, after ocular artifact compensation, will be flagged for rejection. This is probably the most widely used technique for detecting artifacts in ERP data. A check of this type is often applied to a single EOG channel or a frontal EEG channel. In contrast, Artfil checks all EEG channels for absolute voltage artifacts after correcting the EEG for EOG contamination. It does not search the EOG for absolute-voltage artifacts of this type. It does, however, search the EOG for voltages large enough to cause amplifier saturation because amplifier saturation renders ocular-artifact compensation impossible.

4. Large rms voltages are detected by comparing the overall root-mean-squared amplitude of each epoch of data with a criterion value, rmscrit. The quantity compared to rmscrit is the standard deviation of the points within a given epoch and channel. An epoch and channel of data will be flagged for rejection if its rms voltage exceeds rmscrit after ocular artifact compensation has been performed. The rmscrit should be set to a value substantially smaller than the absolute-voltage criterion, voltcrit. This is because overall rms voltages are inherently smaller than peak voltages.

5. Dead EEG channels are detected by comparing the overall rms amplitude of each epoch of data with a criterion value, deadcrit. Any epoch and channel with an overall rms voltage less than deadcrit will be flagged for rejection.

6. Amplifier saturation is detected by comparing the absolute value of each voltage in each epoch of data with a criterion value, clipcrit. For computational speed and convenience, we define clipcrit as an input voltage large enough to cause an amplifier to saturate. That is, the output voltage that will produce clipping is divided by amplifier gain to yield the value of clipcrit. A error in amplifier calibration could cause Artfil to miss instances of clipping. This can be minimized by setting clipcrit to 90-95% of an amplifier's actual clipping voltage, assuming calibration accuracy. Separate clipping criteria can be applied to the EEG and EOG data, as discussed in section 2.

TREATMENT OF OCULAR ARTIFACTS

Eye movements and blinks produce electrical potentials that can be recorded with EEG electrodes. Changes in the spatial distribution of the eyes' standing electrical fields that occur when the eyes move in the head cause eye-movement artifacts. Eyeblink artifacts are transient changes in these fields due to resistance changes associated with eyelid movements (see reference 1 for a discussion). The locations of the active and reference electrodes determine the amplitudes of both types of artifacts. When recorded by EEG electrodes placed at sites commonly used for ERP recording, the amplitudes of ocular artifacts are often larger than those of ERPs.

Generally, eye movements are monitored by recording at least one channel of EOG along with the EEG to detect EEG segments that may be contaminated by ocular potentials. The customary procedure is to reject an epoch of EEG when the EOG exceeds a criterion absolute voltage. This widely used strategy is simple and requires only two assumptions. The first assumption is that the detection procedure is sensitive enough that any contamination produced by undetected eye movements can be safely ignored. The second is that excluding EEG with eye-movement artifacts does not produce a biased data set. To our knowledge, neither assumption has been thoroughly examined and verified.

An alternative is to use EOG recordings to remove ocular potentials from the EEG (see reference 1 for a review), as in Artfil. This approach also involves assumptions, which center on the accuracy with which direct recordings of the EOG can be used to estimate the EOG contamination present in EEG recordings. All compensation procedures of this type involve subtracting suitably scaled EOG waveforms from the EEG waveforms. (Sometimes the waves are decomposed into frequency components, and the different frequencies are scaled separately.) Because the EOG is recorded from the head, EOG electrodes probably always record some EEG activity. Hence, these procedures probably distort ERP data somewhat because they involve subtracting brain activity recorded by the EOG electrodes from brain activity recorded by the EEG electrodes. The magnitude of this problem has not been thoroughly studied. One way to reduce EOG contamination is to record the EOG differentially from a pair (or pairs) of electrodes adjacent enough so that the local EEG is nearly identical in each (3). Differential amplification will then tend to remove the EEG from the EOG recording.

The ocular-artifact compensating routines of Artfil presume that bipolar EOG recordings result from two pairs of electrodes, each containing a recording electrode and a reference electrode. The program assumes that one pair of electrodes obtained vertical EOG data, perhaps from an electrode above one eye referred to an electrode below that eye. These data are used to detect and compensate the data for eyeblink artifacts. The program also assumes a second pair of electrodes obtained horizontal or oblique EOG data. We refer to this electrode pair as the lateral EOG channel. Such data might be recorded from electrodes placed to the left and right of one eye, or obliquely about one eye. Artfil uses these data to correct the EEG for eye movements.

Artifil corrects the EEG for artifacts caused by eye movements and blinks using a variant of the procedure described by Gratton et al. (2). Their algorithm assumes that the waveform recorded from an EEG electrode can be approximated as the sum of two time series. The first time series is the actual EEG waveform; the other is a linearly attenuated version of the EOG.

By this assumption, the actual EEG can be recovered from the recorded EEG by subtracting the appropriately scaled EOG point-by-point.

The basic procedure consists of two steps, each carried out on an epoch-by-epoch basis. In the first step, a least-squares cross regression is calculated using the EEG time series as criterion variables and the EOG as predictor variables. This regression estimates the constant to multiply the EOG time series to get the best linear prediction of the EEG time series. The constant is an estimate of the proportion of the EOG contained in the recorded EEG. Its value is estimated separately for each EEG recording site. In the second step, part of the recorded EOG is subtracted in pointwise fashion from the EEG. The amount subtracted from the EEG is determined by the proportionality constant estimated in step one.

These procedures are different for eye movements and blinks. For eye movements, the proportion is estimated from complete EEG and EOG time series. For blinks, the proportion of EOG subtracted from the EEG is estimated from data obtained during periods in which the local slope of the vertical EOG (calculated in a 10-ms time window) exceeds a criterion value.

When the blink-slope criterion is properly chosen, the data used to estimate the proportionality constant for eyeblink compensation will be selected from the leading and trailing edges of eyeblink-artifact waveforms (i.e., when the eyelids are moving rapidly). Selecting an appropriate value for this criterion is critical to the performance of the algorithm. If the blink-scope criterion is too high, the algorithm either fails to detect blinks or produces unstable estimates of the blink-scaling constant based on small numbers of data points. On the other hand, if the value is too low, the algorithm mistakes EOG noise for blinks and, thereby, underestimates the blink-scaling constant and undercorrects the EEG for the effects of blinks.

We can only advise on how to select a value for the blink criterion. The shapes of eyeblink waveforms depend on where the EOG electrodes are placed and on how the data are filtered. Blinks also vary from one individual to another and from one blink to the next. We set the slope criterion to a value that is less than the slopes of the blink waveform leading edges when blinks are 25-75% of their maximum amplitudes. Because blinks are variable and because the results of different criterion settings must be checked visually blink by blink, selecting a value of the blink criterion can be time-consuming.

The procedure to compensate for ocular artifacts described by Gratton et al. (2) includes a third step that is not included in Artfil. It involves subtracting a signal-average of all epochs of the ERP data (with ocular artifacts included) from each individual epoch of data before estimating the scaling constants and subtracting the scaled EOG from the EEG. This step is intended to remove the brain activity evoked by eyeblinks from the EEG data. We examined the procedure and found that in our hands the ocular-artifact compensating algorithm performed better without it.

2. USING THE PROGRAM

DATA FILES

Artfil requires two input data files and creates two output data files. The main input data file contains digitized EEG and EOG; the default name is **pochchan**. The second input data file contains parameters used by Artfil for artifact detection and EOG artifact compensation. The default name of this file is **artifact.p2**.

The main output data file, the default name of which is **epochchan.c**, contains digitized EEG with the ocular-artifact compensating algorithm applied. Because Artfil does not discard data with uncorrectable artifacts, **epochchan.c** contains as much EEG data as the input file **epochchan**. Artifil writes the epoch and channel numbers of data with uncorrectable artifacts to a second output data file named **artifacts**.

The physiological data in **epochchan** and **epochchan.c** are stored as time series of digitized EEG and EOG amplitudes scaled in 0.1-uV units. Values of the EEG and EOG data are stored in the files, which are direct-access, as two-byte integers. Each time series corresponding to data from one recording channel in one epoch is stored as a separate record.

The layout of the **epochchan** files is best illustrated by considering each voltage in the file as an element in a triply subscripted voltage array, $v(e,c,i)$. The subscript e indexes the ordinal (and temporal) position of the recording epoch from which v was obtained. The value of e varies from 1 to ne, the number of epochs in the data set under consideration. The index c indicates the number of the recording channel from which v was obtained. The value of c varies from 1 to nc, the number of recording channels. The index i refers to the ordinal (and temporal) position of v within the current recording epoch. The value of i varies from 1 to ni. Thus, $v(1,2,3)$ is the third voltage point obtained from channel two in epoch one.

Values of v are organized in the **epochchan** files such that index i varies most rapidly. (Recall that ni voltages from channel c in epoch e comprise one record of **epochchan**.) The channel index, c, varies next most rapidly, and the epoch index, e, varies least rapidly.

The input data file **artifact.p2** contains several parameters that Artfil uses to detect artifacts. The user should tailor the parameters for each recording system and experiment and enter one number per line in the file as ASCII-coded numbers in the order indicated below.

clipcrit - The criterion voltage for detecting EEG amplifier clipping. For simplicity, the value of clipcrit is expressed as the voltage, in uV at the amplifier inputs, sufficient to produce clipping. Any epoch of data in which clipping is detected will not be corrected for ocular artifacts and will be flagged for rejection. A conservative value of clipcrit, assuming EOG amplifier gains of 20,000 and saturation voltages of 5.0 V, would be 225.0. A separate voltage criterion is used to detect EOG amplifier clipping (see clipcrit2 below).

clipcrit2 - A second clipping-criterion voltage that is used to detect EOG amplifier clipping. Again, Artfil detects amplifier clipping by monitoring amplitudes expressed as amplifier-input voltages. The amplifier-input voltages of the EOG, however, are much larger than those of the EEG. Hence, the EOG and EEG amplifiers may be operated at different gains. If the EOG and EEG amplifiers differ in gain settings (but are otherwise similar) they will saturate at different input voltages.

A solution to this problem is to define separate clipping criteria for the EEG and EOG amplifiers: Artfil uses the value of clipcrit2 to detect EOG amplifier clipping. Any epoch in which the absolute voltage of the EOG exceeds clipcrit2 will not be used in ocular-artifact correction and will be flagged for rejection. A conservative value of clipcrit2, assuming EOG amplifier gains of 2000 and output saturation voltages of 5.0 V, would be 2250.0.

blinkcrit - A criterion for detecting blinks. After the method of Gratton et al. (2), blinkcrit is defined as a criterion value of the local slope of the vertical EOG trace measured in a 10-ms time interval. Artfil adjusts the input value of blinkcrit linearly to accommodate sampling intervals that differ from 10 ms. As discussed previously, determining an appropriate value for blinkcrit may require some experimentation.

spikecrit - A rate of voltage change, in uV/ms, that identifies the presence of a high-frequency spike in an EEG channel. The criterion is applied to differences calculated between successive voltages in each epoch and channel of data except for the vertical EOG. As noted, the EOG is not scanned for spikes for fear of confusing them with blinks. A suggested value of spikecrit is 33.0.

voltcrit - An absolute-voltage criterion, expressed in uV, used for rejecting epochs of data. Any epoch and channel of EEG data that contains an absolute voltage exceeding voltcrit after the ocular artifact compensation procedure has been applied will be flagged for rejection. A suggested value of voltcrit is 40 uV.

rmscrit - A root-mean-squared voltage criterion used for rejecting epochs of data (scaled in uV). The standard deviation of the points within a given epoch and channel are compared to rmscrit. An epoch and channel of data will be flagged for rejection if its rms voltage exceeds rmscrit after ocular artifact compensation has been performed.

deadcrit - A root-mean-squared voltage criterion used to identify dead amplifiers (scaled in uV). Any epoch and channel with an overall rms voltage less than deadcrit will be flagged for rejection. We suggest a value near 4.0 uV.

samprate - The EEG and EOG sampling rate scaled in data points per second. Artfil uses samprate when applying the EOG slope criterion (blinkcrit) to the data. This variable is an integer.

The output file **artifacts** is a summary of the artifacts found by Artfil's artifact detecting algorithms. The file is written to the directory in which the program found the original **epochchan** file. the file contains one record of artifact data for each epoch and channel of data examined. As in **epochchan** files, data records for different channels are written in ordered blocks.

Each block of records contains the data from nc channels for a single epoch. Each record within a block corresponds to the data from one channel.

Each analytical program uses the information in **artifacts** differently. Generally, a program opens **epochchan.c** and **artifacts** files simultaneously. It then reads the information in the kth record of Artifacts, which is the summary of artifacts for the physiological data stored in record k of **epochchan.c**. The program uses this information to determine whether the epoch of data in record k of **epochchan.c** is usable. If so, the program reads the physiological data and analyzes it. If not, the program either proceeds to another record or exits as appropriate. Each record in **artifacts** contains ten, 2-byte-integer data fields described below.

Field	Interpretation
1.	1 if a clip or spike was detected in the EOG in the current epoch, or if a clip, spike, rms, or absolute-voltage artifact was detected in the EEG in the current epoch; 0 otherwise.
2.	The number of the current epoch.
3.	The number of the current channel.
4.	1 if the current channel is the vertical EOG and a blink was detected in the current epoch; 0 otherwise.
5.	1 if the current channel is an EEG channel and an absolute-voltage artifact was detected in the current epoch; 0 otherwise.
6.	1 if the current channel is an EEG channel and an rms artifact was detected in the current epoch; 0 otherwise.
7.	1 if the current channel is dead in the current epoch; 0 otherwise.
8.	1 if the current channel is not the vertical EOG channel and a spike artifact was found in the current epoch; 0 otherwise.
9.	1 if a clipping artifact was found in the current channel and epoch; 0 otherwise.
10.	The overall rms amplitude of the current channel and epoch, rounded to the nearest uV.

RUNNING THE PROGRAM

Artfil allows the user to specify several arguments on the command line when the program is loaded. To run Artfil, the user enters:

```
programname  ARGUMENTS
```

where programname is the name assigned to the executable version of the program, and ARGUMENTS is a sequence of command line arguments. Only the first two arguments (-c and -n) are required; the rest are optional. The list of valid command line arguments is:

-c<number of channels>, required
The number of channels of data in the **epochchan** file.

-n<number of points per epoch>, required
The number of data points per channel per epoch.

[-v<vertical eye channel>], default = 1
The channel number of the vertical EOG channel if not channel 1.

[-l<lateral eye channel>], default = 2
The channel number of the lateral EOG channel if not channel 2.

[-F<first epoch to process>], default = 1
The first epoch processed in the current run. If too large to process at one time, an **epochchan** file can be processed as blocks of epochs in sequential runs. The first and last epoch of each run indicated with this argument and the -L argument. If not specified, the first epoch processed will be epoch 1.

[-L<last epoch to process>], default = 10
The last epoch processed in the current run. This argument is used in conjunction with the -F argument. If not specified, the last epoch processed will be epoch 10.

[-e(chan. to omit from artifact filtering)], default = no omissions
Indicates which channel of data should not be processed by the artifact filter. Data in any channel so indicated will be written to the output file **epochchan.c** exactly as it was read from the input file **epochchan**. The argument may be listed more than once on a given command line.

[-m (if present, perform three-point filtering on data)], default = none
Used for three-point smoothing of the input data.

[-V(verbose)], default = no verbose
If this argument is present, Artfil will print numerous messages pertinent to the status of the analysis.

[-P<artifact parameter file path name>], default = artifact.p2
Used if the artifact parameter file is not the default. Enter the new file name.

[-I<input file path name>], default = epochchan
Used when the input physiological data file is not the default. Enter the new file name.

[-O<output file path name>], default = epochchan.c
Used when the output physiological data file is not the default. Enter the new file name.

[-A<artifact output file path name>], default = artifacts
Used when the artifact summary file is not the default. Enter the new file name.

For example, entering 'artfil' will display a help screen containing a list of command line options. If we then enter:

```
artfil -c10 -n250 -v1 -12 -F125 -L250 -e10
```

the program will read 10 channels of data (-c10) comprising 250 data points per channel per epoch (-n250). The program will treat channel 1 as the vertical EOG (-v1) and channel 2 as the lateral EOG (-12). Artfil will analyze epoch 125 (-F125) first and epoch 250 (-L250) last but will not examine data in channel 10 (-e10). If no data files are specified on the command line, the default files will be used by the program.

3. SYSTEM REQUIREMENTS

DATA LIMITATIONS

For efficiency on the MASSCOMP 5500, the maximum number of channels is 20, the maximum number of epochs per run is 350, and the maximum number of points per channel per epoch is 400. These values were set to allow Artfil to run entirely with nonvirtual arrays. They can be modified depending on available memory. To adjust the maximum channels, epochs, or points, the following parameter statements can be edited in Artfil's source code:

```
parameter    (MAXCHANNELS =   20)    ! Maximum number of channels
parameter    (MAXEPOCHS   =  350)    ! Maximum number of epochs
parameter    (MAXPOINTS   =  400)    ! Maximum number of points per
                                       epoch
```

COMPATIBILITY

Artfil runs on a MASSCOMP 5500 computer under MASSCOMP Real-Time Unix Version 4.0. It is written in Fortran 77 with DEC VAX extensions, and it should be reasonably compatible with most compilers on VAX, Hewlett-Packard, IBM, and IBM PC-compatible computers.

The major departure from standard Fortran 77 in Artfil is the use of do-loops that end with "enddo" statements. Some compilers do not support control structures of this type. They can be replaced easily with traditional loops. Another difference from standard Fortran 77 is the use of command line arguments in Artfil. If a compiler does not support command line arguments, Artfil can be modified to read the argument values from a file.

Lastly, before the variable declaration statements, Artifil uses an "implicit none" statement, which requires that all variables be explicitly declared. It can be removed. Other incompatibilities may occur in input and output statements, which are notoriously nonstandard across Fortran compilers.

REFERENCES

1. Barlow, J.S., "Artifact Processing (Rejection and Minimization) in EEG Data Processing." In F.H. Lopes da Silva, W. Storm van Leewen, and A. Remond (Eds.), Clinical Applications of Computer Analysis of EEG and other Neurophysiological Signals, Elsevier Press, New York, NY, 1986, pp. 15-64.

2. Gratton, G., Coles, G.H., and Donchin, E., "A New Method for Off-Line Removal of Ocular Artifacts." Electroencephalography and Clinical Neurophysiology, Vol. 55, pp. 468-484, 1983.

3. Stanny, R.R., Mapping the Event-related Potentials of the Brain: Theoretical Issues, Technical Considerations, and Computer Programs, NAMRL SR 88-1, Naval Aerospace Medical Research Laboratory, Pensacola, FL, October, 1988.

APPENDIX

```
      program artfil
c
c Author:    Dr. Robert R. Stanny (NAMRL), September 1986
c
c Modification Log:
c
c    1.  November 1987, Sam J. La Cour, Jr. (NARDAC)
c
c        Made the program compatible with the files and methods used in
c        data acquisition programs written for the system circa 1987.
c
c    2.  January 1989, Sam J. La Cour, Jr. (NARDAC)
c
c        Involved allowing specification of eye channels
c        and changing the way in which blinks are scaled
c        prior to calculation of scaling constant.
c
c    3.  September 1989, Sam J. La Cour, Jr. (NARDAC)
c
c        Place "clipcrit2" variable in parameter file after "clipcrit".
c
c    4.  January 1990, Sam J. La Cour, Jr. (NARDAC)
c
c        Modified calculation of adjusted blink and spike criteria.
c
c End Modifications
c
c Program Usage:
c
c 1.  Command Line:
c
c     artfil -c<number of channels>, required
c             [-v<vertical eye channel>], default 1
c             [-l<lateral  eye channel>], default 2
c             [-F<first epoch to process>], default 1
c             [-L<last  epoch to process>], default 10
c             [-e<channel to omit from artifact filtering>],
c                 default no omissions
c             [-m (if present, perform 3 point filtering on data)],
c                 default none
c             [-V (verbose)], default no verbose
c             [-P<artifact parameter file path name>],
c                 default "artifact.p2"
c             [-I<input file path name>], default "epochchan"
c             [-O<output file path name>], default "epochchan.c"
c             [-A<artifact output file path name>],
c                 default "artifacts"
c
c 2.  The program assumes the existance of "epochchan" in the current
c     directory.  This file contains the EOG and EEG data.  Also, the
c     file "epochinfo" must exist in the same directory.  It contains
c     the number of epochs (first line) and the number of points per
```

```
c     epoch (second line).  This is a text file.
c
c 3.  The program reads the following parameters from the artifact
c     parameter file:
c
c     clipcrit  uV voltage used to detect amplifier clipping in EEG channels
c     clipcrit2 uV voltage used to detect amplifier clipping in EOG channels
c     blinkcrit uV/10ms voltage change used as a criterion for
c               identifying blinks in eye channels
c     spikecrit uV/1ms voltage change used as a criterion for
c               identifying spikes in EEG channels
c     voltcrit  Absolute voltage for identifying an outlying data point
c     rmscrit   The critical epochwise rms for identifying a noisy channel
c     deadcrit  The minimum epochwise rms for identifying a dead EEG channel
c     samprate  The integer EEG sampling rate (points / sec)
c
c The file is in the format of one variable per line, free format.
c
c Important Program Variables:
c
c v1(p) =     Voltage at time point p, of a vector of EEG data (or sometimes
c             EOG data).  The vector contains real values in microvolts.
c             The points are organized in temporal order from the first point
c             of the epoch to the last point of the epoch.
c v2(p) =     Voltage at point p of the vertical EOG.  Organization
c             of the vector is like that of v1.
c v3(p) =     Same as v2 for the lateral eog.
c nptot =     The total number of data points per channel (= np * ne).
c nc =        # of channels.
c ne =        Epochs of data per channel.
c np =        Points/epoch/channel.
c LEYECHAN =  Integer channel number of the lateral EOG data.
c VEYECHAN =  Integer channel number of the vertical EOG data.
c nptot =     The number of epochs actually analyzed * points/epoch.
c
c Input Data File Structure:
c
c    The EEG and EOG data files should be direct-access files containing
c    a vector of two-byte integer (integer*2) values in each record.  The
c    record organization is:
c
c        epoch 1 channel 1 vector
c        epoch 1 channel 2 vector
c           .
c           .
c           .
c        epoch 1 channel n vector
c        epoch 2 channel 1 vector
c        epoch 2 channel 2 vector
c           .
c           .
c           .
c        epoch N channel n vector
c
c Notes:
```

```
c
c     The number of channels cannot exceed 20, the maximum number of epochs
c     cannot exceed 700 and the maximum points per epoch cannot exceed 400.
c     These are limitations due to the largest arrays we can run on our computer
c     and could be increased on other machines with more memory.
c
c Variable declarations:
c
        implicit none
c
        integer MAXCHANNELS,MAXEPOCHS
        integer NETOT,MAXPOINTS, VECTORSIZE
        integer ARTFILE,INFILE,OUTFILE,PARMFILE
        integer INFOFILE
        integer STDIN, STDOUT, STDERR
c
        parameter (STDIN        =       5)      ! Standard input
        parameter (STDOUT       =       6)      ! Standard output
        parameter (STDERR       =       6)      ! Standard error (usually 0)
        parameter (MAXCHANNELS  =       15)     ! Maximum number of channels
        parameter (MAXEPOCHS    =       700)    ! Maximum number of epochs
        parameter (NETOT        =       700)    ! Ditto
        parameter (MAXPOINTS    =       400)    ! Maximum number of points
c
c May have to use some sort of dynamic scheme if the following get too big
c (or just run the 1/2 epochs at a time, which is fine for an epochwise
c algorithm)
c
        parameter (VECTORSIZE   =       MAXPOINTS * MAXEPOCHS)
c
        parameter (ARTFILE      =       10)
        parameter (INFILE       =       11)
        parameter (OUTFILE      =       12)
        parameter (PARMFILE     =       13)
        parameter (INFOFILE     =       14)
c
c Switches
c
        logical verbose,                ! Verbosity of displayed output
     .          filter,                 ! Subject the data to 3 point filter
     .          eegchan(MAXCHANNELS)    ! Indicates if the channel is to be
                                        ! artifact filtered
c
        integer
     .          i,j,k,
     .          nn,             ! Statistics counter
     .          ios,            ! General purpose error return
     .          reclen,         ! Epochchan record length in bytes
     .          recno,          ! Current record number
     .          VEYECHAN,       ! Vertical eye channel
     .          LEYECHAN,       ! Lateral eye channel
     .          nctot,          ! Total number of channels
     .          nc,             ! Ditto, in another context
     .          np,             ! Number of points/epoch
     .          ichan,          ! Channel index
```

```
     .          iepoch,          ! Epoch index
     .          ipoint,          ! Sampled point index
     .          nepochs,         ! Number of epochs in a run
     .          firstepoch,      ! First epoch to process
     .          lastepoch,       ! Last epoch to process
     .          delta,           ! Actual sampling increment used
     .          n,               ! General purpose counter
     .          ival,            ! Integer command line argument value
     .          getcwd,          ! System call to get current working directory
     .          retval,          ! General purpose return value
     .          nominalfirst,    ! Nominal (relative) first epoch number
     .          nominallast,     ! Nominal (relative) last epoch number
     .          nominale,        ! Nominal (relative) current epoch number
     .          nptot,           ! Total number of points in a sample
     .          pass,            ! Used for controlling artifact filtering loop
     .          firstpoint,      ! First point to use in a v_() array
     .          lastpoint,       ! Last point to use in a v_() array
     .          ptemp,           ! Used to sequentially index through
     .                           ! v_() arrays
     .          lencurrentpath,  ! Length of constructed current path
     .          leninpath,       ! Length of constructed input file path
     .          lenparmpath,     ! Length of output parameter file path
     .          leninfopath,     ! Length of constructed information file path
     .          lenartpath,      ! Length of artifact file path
     .          lenoutpath       ! Length of constructed output file path
c
      integer
     .          blink(MAXCHANNELS,MAXEPOCHS),     ! Blink indicator
     .          voltart(MAXCHANNELS,MAXEPOCHS),   ! Absolute voltage indicator
     .          clipart(MAXCHANNELS,MAXEPOCHS),   ! Voltage clipping indicator
     .          rmsart(MAXCHANNELS,MAXEPOCHS),    ! Excessive RMS indicator
     .          deadchan(MAXCHANNELS,MAXEPOCHS),  ! Not enough RMS indicator
     .          spike(MAXCHANNELS,MAXEPOCHS),     ! Voltage spike indicator
     .          sumart(MAXCHANNELS,MAXEPOCHS)     ! Artifact summary variable
c
      integer*2
     .          dummy(MAXPOINTS),        ! Used to read input data
     .          arts(10)                 ! Artifact indicator array
c
      real
     .          v1(VECTORSIZE),          ! Vector used to hold EEG
     .          v2(VECTORSIZE),          ! Lateral EOG vector
     .          v3(VECTORSIZE),          ! Vertical EOG vector
     .          rdummy(MAXPOINTS)        ! Input data converted to float
c
      real
     .          rms(MAXCHANNELS,MAXEPOCHS),      ! RMS (per channel and epoch)
     .          mean(MAXCHANNELS,MAXEPOCHS)      ! Mean voltage (per channel and epoch)
c
      real
     .          voltcrit,                ! Absolute voltage criterion
     .          blinkcrit,               ! Blink rate of change criterion
     .          spikecrit,               ! Spike rate of change criterion
     .          rmscrit,                 ! Excessive RMS criterion
     .          deadcrit,                ! Dead channel criterion
```

```
     .          clipcrit,                ! Clipping criterion

     .          clipcrit2,               ! Adjusted eog clipping criterion
     .          cov,                     ! Calculated covariance
     .          ss,                      ! Sum of squares
     .          tx,                      ! Sum of x
     .          ty,                      ! Sum of y
     .          v1adj,                   ! Value of v1 with any corrections applied
     .          v2adj,                   ! Value of v2 with any corrections applied
     .          rval,                    ! Real command line value
     .          b(MAXCHANNELS),          ! Slope of correction line for a given channel
     .          samprate,                ! Sampling rate
     .          d,                       ! Temporary variable
     .          realpoints,              ! Float(nptot)
     .          realint,                 ! Actual sampling interval
     .          realn                    ! Float(number of points checked per channel)
c
      character*256
     .          arg,                     ! Command line string
     .          cval                     ! Character command line value
c
      character*1
     .          option,                  ! "transparm" returned option letter
     .          optiontype               ! "transparm" returned option type
c
      character*256
     .          currentpath,             ! Current directory path
     .          artpath,                 ! Artifact file path (artifacts)
     .          inpath,                  ! Input file path (epochchan)
     .          outpath,                 ! Output file path (epochchan.c)
     .          parmpath,                ! Parameter file path (artifact.p2)
     .          infopath                 ! Path to information file (epochinfo)
c
c Initialize command line variables with default values
c
      n = 0
      nc = 0
      np = 0
      currentpath = ' '
      inpath = 'epochchan'
      call removespaces (inpath,leninpath)
      outpath = 'epochchan.c'
      call removespaces (outpath,lenoutpath)
      parmpath = 'artifact.p2'
      call removespaces (parmpath,lenparmpath)
      artpath = 'artifacts'
      call removespaces (artpath,lenartpath)
      VEYECHAN = 1
      LEYECHAN = 2
      filter = .false.
      do i= 1, MAXCHANNELS
          eegchan(i) = .true.
      enddo
      firstepoch = 1
      lastepoch = 10
```

```
        verbose = .false.
c
c Process command line arguments
c
1001    n = n + 1
        call getarg (n,arg)
        if (arg.eq.' ') goto 1002
        call transparm (arg,option,optiontype,ival,rval,cval)
c
        if (option.eq.'c') then
            nc = ival
            nctot = nc
        else if (option.eq.'v') then
            VEYECHAN = ival
        else if (option.eq.'l') then
            LEYECHAN = ival
        else if (option.eq.'F') then
            firstepoch = ival
        else if (option.eq.'L') then
            lastepoch = ival
        else if (option.eq.'e') then
            eegchan(ival) = .false.
        else if (option.eq.'m') then
            filter = .true.
        else if (option.eq.'V') then
            verbose = .true.
        else if (option.eq.'I') then
            inpath = cval
            call removespaces (inpath,leninpath)
        else if (option.eq.'O') then
            outpath = cval
            call removespaces (outpath,lenoutpath)
        else if (option.eq.'P') then
            parmpath = cval
            call removespaces (parmpath,lenparmpath)
        else if (option.eq.'A') then
            artpath = cval
            call removespaces (artpath,lenartpath)
        endif
        goto 1001
1002    continue
c
c If no parameters, display help and exit
c
        if (n.eq.1) then
            write (STDOUT,*) 'artfil: Evoked potential artifact detector'
            write (STDOUT,*) '        and filter -- Single epoch corrections'
            write (STDOUT,*)
            write (STDOUT,*) '  -c<number of channels>, required'
            write (STDOUT,*) '  [-v<vertical eye channel>], default 1'
            write (STDOUT,*) '  [-l<lateral eye channel>], default 2'
            write (STDOUT,*) '  [-F<first epoch to process>], default 1'
            write (STDOUT,*) '  [-L<last  epoch to process>], default 10'
            write (STDOUT,*) '  [-e(channel to omit from artifact filtering)],'
            write (STDOUT,*) '      default no omissions'
```

```
            write (STDOUT,*) '  [-m(perform 3 point filtering on data)],'
            write (STDOUT,*) '      default no filtering'
            write (STDOUT,*) '  [-V(verbose)], default no verbose'
            write (STDOUT,*) '  [-P<artifact parameter file path name>],'
            write (STDOUT,*) '      default "artifact.p2"'
            write (STDOUT,*) '  [-I<input file path name>],'
            write (STDOUT,*) '      default "epochchan"'
            write (STDOUT,*) '  [-O<output file path name>],'
            write (STDOUT,*) '      default "epochchan.c"'
            write (STDOUT,*) '  [-A<artifact output file path name>],'
            write (STDOUT,*) '      default "artifacts"'
            goto 32767
        endif
c
c Setup the file path information
c
        retval = getcwd (currentpath)
        call removespaces (currentpath,lencurrentpath)
c
        i = 0
        if (nctot.le.0) then
            write (STDERR,*) 'artfil: Number of channels must not be ',
     .                       'zero or less'
            i = 1
        endif
        if (i.ne.0) then
            goto 32767
        endif
c
        infopath = 'epochinfo'
        call removespaces (infopath,leninfopath)
c
c Get number and length of epochs from information file
c
        open (unit=INFOFILE, file=infopath(1:leninfopath),
     .        status='old', iostat=ios)
        if (ios.ne.0) then
            write (STDERR,*) 'artfil: Error opening epochinfo, ',
     .                       'iostat is ',ios
            goto 32767
        endif
        rewind (INFOFILE)

        read (INFOFILE,*) nepochs       ! number of epochs
        read (INFOFILE,*) np            ! number of points per epoch
        close (INFOFILE)
c
        lastepoch = nepochs
        if (verbose) then
            write (STDOUT,*) 'Number of epochs: ',nepochs
            write (STDOUT,*) 'Number of points/epoch: ',np
        endif
c
c Read the artifact parameter file:
c
```

```
      open (PARMFILE, file=parmpath(1:lenparmpath),
     .      status='old', iostat=ios)
      if (ios.ne.0) then
          write (STDERR,*) 'artfil: Error opening parameter file, ',
     .                     'iostat is ',ios
          goto 32767
      endif
      rewind (PARMFILE)
      read (PARMFILE,*) blinkcrit
      read (PARMFILE,*) spikecrit
      read (PARMFILE,*) voltcrit
      read (PARMFILE,*) rmscrit
      read (PARMFILE,*) deadcrit
      read (PARMFILE,*) clipcrit
      read (PARMFILE,*) clipcrit2
      read (PARMFILE,*) samprate
      close (PARMFILE)

      if (verbose) then
          write (STDOUT,*) 'Blink criteria is ',blinkcrit,' uV/10ms'
          write (STDOUT,*) 'Spike criteria is ',spikecrit,' uV/ms'
          write (STDOUT,*) 'Volt criteria is  ',voltcrit,' uV'
          write (STDOUT,*) 'RMS criteria is   ',rmscrit,' uV'
          write (STDOUT,*) 'Dead criteria is  ',deadcrit,' uV'
          write (STDOUT,*) 'EEG clipping criteria is ',clipcrit,' uV'
          write (STDOUT,*) 'EOG clipping criteria is ',clipcrit2, ' uV'
          write (STDOUT,*) 'Sampling rate is  ',samprate,' Hz'
      endif
      if (verbose) then
          write (STDOUT,*) 'Vertical eye channel is ',VEYECHAN
          write (STDOUT,*) 'Lateral eye  channel is ',LEYECHAN
      endif
c
      if (verbose) then
          write (STDOUT,*) 'Gain of vertical eye channel is ',
     .        clipcrit2/clipcrit,' % re other eeg channels'
      endif
c
c Initialize data arrays:
c
      do ichan = 1, nc
          do iepoch = firstepoch, lastepoch
              blink(ichan,iepoch) = 0
              voltart(ichan,iepoch) = 0
              clipart(ichan,iepoch) = 0
              rms(ichan,iepoch) = 0.
              rmsart(ichan,iepoch) = 0
              mean(ichan,iepoch) = 0.
              deadchan(ichan,iepoch) = 0
              spike(ichan,iepoch) = 0
          enddo
      enddo
c
c DEFINE SOME VARIABLES:
c
```

```
c Nominalfirst and nominallast are nominal epoch numbers.
c Nominal first is always 1.
c This is the first value in the sequence bounded by
c firstepoch and lastepoch.  nominallast is the ordinal value
c of the last epoch in the sequence.  Nptot is the rank of the
c v1-v3 vectors.
c
        nominalfirst = 1
        nominallast = lastepoch - firstepoch + 1
        nptot = np * (lastepoch - firstepoch + 1)

        if (nptot.gt.VECTORSIZE) then
            write (STDERR,*) 'artfil: Too many epochs, rerun with fewer'
            goto 32767
        endif
c
        reclen = np * 2
c
        if (verbose) then
            write (STDOUT,*) 'First, last epochs:             ',
     .          firstepoch, lastepoch
            write (STDOUT,*) 'Nominal first, last epochs:     ',
     .          nominalfirst, nominallast
            write (STDOUT,*) 'Number of points total (nptot): ',
     .          nptot
            write (STDOUT,*) 'Number of points allowed:       ',
     .          VECTORSIZE
            write (STDOUT,*) 'Number of points/epoch: ',
     .          np
        endif
c
c Correct blink and spike voltage-change criteria for
c quantization errors due to the obtained intervals
c between time points.
c
        realpoints = .01 * float(samprate)
c
c Set the calculating interval in points to the value nearest
c the number of points giving 10 ms.
c
        delta = realpoints + 0.5
        if (delta.lt.1) then
            delta = 1
        endif
        realint = float(delta)/float(samprate)  ! gives seconds
                                                ! corresponding to delta
        blinkcrit = blinkcrit * realint / .01   ! adjust blinkcrit by
                                                ! ratio of delta to 10 ms
        spikecrit = spikecrit * 1000./float(samprate)
        if (verbose) then
            write (STDOUT,*) 'Adjusted blink criteria is ',
     .          blinkcrit,' uV/sample_interval'
            write (STDOUT,*) 'Adjusted spike criteria is ',
     .          spikecrit,' uV/pt'
        endif
```

```
c
c Now, calculate "realn", the number of points to be checked per channel,
c expressed as a floating point number
c
        realn = float(np - 2 * delta)
        if (verbose) then
            write (STDOUT,*) 'The number of points checked per chan. is ',
     .          realn
        endif
c
c Open the input and output files
c
        open (INFILE, file=inpath(1:leninpath), status='old',
     .      access='dir', recl=reclen, iostat=ios)
        if (ios.ne.0) then
            write (STDERR,*) 'artfil: Error opening input file, ',
     .                       'iostat is ',ios
            goto 32767
        endif
        open (OUTFILE, file=outpath(1:lenoutpath), status='unk',
     .      access='dir', recl=reclen)
        if (ios.ne.0) then
            write (STDERR,*) 'artfil: Error opening output file, ',
     .                       'iostat is ',ios
            goto 32767
        endif
c
c We need the eye movement data early on, so we
c get the vertical and horizontal eye data now
c
c First, the vertical eye channel...
c
        if (verbose) then
            write (STDOUT,*) 'Reading vertical eye channel data...'
        endif
        ichan = VEYECHAN
        ptemp = 0
        do iepoch = firstepoch, lastepoch
            recno = nctot * (iepoch - 1) + ichan
            call quickio (INFILE, inpath(1:leninpath), 'old', 'read', recno,
     .          dummy, np, ios)
            if (ios.ne.0) then
                write (STDERR,*) 'artfil: Error reading input file, ',
     .                           'iostat is ',ios
                goto 32767
            endif
c
c Note that the filtering subprogram "smooth3p" scales everything down by a
c factor of 10, to account for the assumption that the input file is in 1/10 uV
c units (i.e. 10.6 uV is represented by 106 in the file).  If filtering is not
c performed, then we need to adjust the values ourselves.
c
            if (filter) then
                call smooth3p(dummy,rdummy,np)
            else
```

```
                do i = 1, np
                    rdummy(i) = dummy(i) / 10.0
                enddo
            endif
            do ipoint = 1, np
                ptemp = ptemp + 1
                v2(ptemp) = rdummy(ipoint)
            enddo
        enddo
c
c And then the lateral eye channel
c
        if (verbose) then
            write (STDOUT,*) 'Reading lateral eye channel data...'
        endif
        ichan = LEYECHAN
        ptemp = 0
        do iepoch = firstepoch, lastepoch
            recno = nctot * (iepoch - 1) + ichan
            call quickio (INFILE, inpath(1:leninpath), 'old', 'read',
     .          recno, dummy, np, ios)
            if (ios.ne.0) then
                write (STDERR,*) 'artfil: Error reading lateral eye ',
     .                           'channel data, iostat is ',ios
                goto 32767
            endif
            if (filter) then
                call smooth3p(dummy,rdummy,np)
            else
                do i = 1, np
                    rdummy(i) = dummy(i) / 10.0
                enddo
            endif
            do ipoint = 1, np
                ptemp = ptemp + 1
                v3(ptemp) = rdummy(ipoint)
            enddo
        enddo
c
c DO A PRELIMINARY CHECK OF THE VERTICAL EOG:
c
c Before doing anything else, check the vertical eog channel for
c blinks and clipping.  Do not check it for spikes, in order not to
c confuse blinks with spikes.
c
        if (verbose) then
            write (STDOUT,*) 'Checking vertical EOG for blinks and clipping...'
        endif
        iepoch = firstepoch - 1
        do nominale = nominalfirst, nominallast
            iepoch = iepoch + 1
            j = 0
            k = 0
            d = 0.0
            firstpoint = (nominale - 1) * np + 1
```

```
            lastpoint = firstpoint + np - 1
            do ipoint = firstpoint, lastpoint-delta
                if (abs(v2(ipoint)-v2(ipoint+delta)).gt. blinkcrit) then
                    if (j.eq.0) then
                        j = ipoint
                        d = abs(v2(ipoint)-v2(ipoint+delta))
                    endif
                    blink(VEYECHAN,iepoch) = 1
                endif
                if (abs(v2(ipoint)).gt.clipcrit2) then
                    if (k.eq.0) then
                        k = ipoint
                    endif
                    clipart(VEYECHAN,iepoch) = 1
                endif
            enddo
            if (verbose.and.blink(VEYECHAN,iepoch).eq.1) then
                print *,'Epoch ',nominale,' blink starting at ',
     .              j-firstpoint,' Vdiff=',d
            endif
            if (verbose.and.clipart(VEYECHAN,iepoch).eq.1) then
                print *,'Epoch ',nominale,' clipped at pt ',
     .              k-firstpoint, ' voltage is ',v2(k)
            endif
        enddo
c
c First, write vertical EOG directly to output file
c
        if (verbose) then
            write (STDOUT,*) 'Writing Veog to output...'
        endif
        ichan = VEYECHAN
        do iepoch = firstepoch, lastepoch
            recno = nctot * (iepoch - 1) + ichan
            call quickio (INFILE, inpath(1:leninpath), 'old', 'read', recno,
     .          dummy, np, ios)
            if (ios.ne.0) then
                write (STDERR,*) 'artfil: Error reading vertical EOG ',
     .                           'data, iostat is ',ios
                goto 32767
            endif
            call quickio (OUTFILE, outpath(1:lenoutpath),'unk','write',recno,
     .          dummy, np, ios)
            if (ios.ne.0) then
                write (STDERR,*) 'artfil: Error writing vertical EOG ',
     .                           'data, iostat is ',ios
                goto 32767
            endif
        enddo
c
c Now, process remaining channels (Lateral eog and all eeg)
c
        if (verbose) then
            write (STDOUT,*) 'Processing remaining channels...'
        endif
```

```
c
        pass = 0
        ichan = 0
c
150     continue        ! Top of loop through channels
c
c First, check and correct the lateral eye channel.  From that point on,
c process the eeg channels in order.  The vertical eye channel is not
c checked.
c
        if (pass.eq.0) then
            ichan = LEYECHAN
            pass = pass + 1
        else if (pass.eq.1) then
            ichan = 3
            pass = pass + 1
        else
            ichan = ichan + 1
            pass = pass + 1
        endif
c
        if (ichan.gt.nc) then
            goto 3100
        endif
c
c Skip the non-eeg channels, but write the data to the output file
c unchanged
c
        if (.not.eegchan(ichan)) then
            do iepoch = firstepoch, lastepoch
                recno = nctot * (iepoch - 1) + ichan
                call quickio (INFILE, inpath(1:leninpath), 'old', 'read',
     .              recno, dummy, np, ios)
                if (ios.ne.0) then
                    write (STDERR,*) 'artfil: Error reading non-EEG ',
     .                          'data, iostat is ',ios
                    goto 32767
                endif
                call quickio (OUTFILE, outpath(1:lenoutpath),'unk','write',
     .              recno, dummy, np, ios)
                if (ios.ne.0) then
                    write (STDERR,*) 'artfil: Error writing non-EEG ',
     .                          'data, iostat is ',ios
                    goto 32767
                endif
            enddo
            goto 150
        endif
c
c If we are analyzing EOG data, we read the EOG into
c v1 from v3.  Otherwise we get the EEG from file.
c Remember: v1 is the work vector, v2 is the Veog vector,
c           and v3 is the Leog vector
c
        if (ichan.eq.VEYECHAN) then
```

```
            do i = 1, nptot
                vl(i) = v2(i)
            enddo
        else if (ichan.eq.LEYECHAN) then
            do i = 1, nptot
                vl(i) = v3(i)
            enddo
        else
            ptemp = 0
            do iepoch = firstepoch, lastepoch
                recno = nctot * (iepoch - 1) + ichan
                call quickio (INFILE, inpath(1:leninpath), 'old',
     .              'read', recno, dummy, np, ios)
                if (ios.ne.0) then
                    write (STDERR,*) 'artfil: Error reading EEG ',
     .                          'data, iostat is ',ios
                    goto 32767
                endif
                if (filter) then
                    call smooth3p(dummy,rdummy,np)
                else
                    do i = 1, np
                        rdummy(i) = dummy(i) / 10.0
                    enddo
                endif
                do ipoint = 1, np
                        ptemp = ptemp + 1
                        vl(ptemp) = rdummy(ipoint)
                enddo
            enddo
        endif
c*******************************************************************************
c
c BEGIN THE PRE-CORRECTION ARTIFACT CHECK.
c
c A final check for absolute voltage artifacts is performed after lateral
c eye movement artifacts are removed from the eog, in the eye-artifact
c filter section below
c
        iepoch = firstepoch - 1
        do nominale = nominalfirst, nominallast
            iepoch = iepoch + 1
            firstpoint = (nominale - 1) * np + 1
            lastpoint = firstpoint + np - 1
            do ipoint = firstpoint+ delta, lastpoint-delta
                rms(ichan,iepoch) = rms(ichan,iepoch)+vl(ipoint)**2.
                mean(ichan,iepoch) = mean(ichan,iepoch)+vl(ipoint)
c
c Spike detection:
c
                if (ichan.ne.VEYECHAN) then
                    if ((vl(ipoint)-vl(ipoint+1)) .gt. spikecrit) then
                        spike(ichan,iepoch) = 1
                    endif
c
```

```
c Voltage clipping detection: Notice that there are separate criteria
c                             for the eye and frontal channels.
c
                    if ((ichan.eq.VEYECHAN).or.(ichan.eq.LEYECHAN)) then
                        if (abs(v1(ipoint)).gt.clipcrit2) then
                            clipart(ichan,iepoch) = 1
                        endif
                    else        ! other channels (normal EEG channels)
                        if (abs(v1(ipoint)).gt.clipcrit) then
                            clipart(ichan,iepoch) = 1
                            if (verbose) then
                                print *,'Epoch ',iepoch,' channel ',ichan,
     .                              ' clipping at pt ',ipoint,' v=',
     .                              v1(ipoint)
                            endif
                        endif
                    endif
                endif
            enddo
c
c Finish calculating epoch means and mean squared values:
c
            rms(ichan,iepoch) = (rms(ichan,iepoch)-
     .          mean(ichan,iepoch)**2.0 / realn) / realn
            mean(ichan,iepoch) = mean(ichan,iepoch)/realn
c
c Try to detect possible rounding errors or precision problems.  Do so
c without stopping the program, but display an error message so the problem
c can be investigated.
c
            if (rms(ichan,iepoch).gt.0.0) then
                rms(ichan,iepoch) = sqrt(rms(ichan,iepoch))
            else
                write (STDERR,*) 'artfil: RMS for epoch ',iepoch,
     .                           'channel ',ichan,
     .                           ' was bad (sumsq < sum**2/n). ',
     .                           ' Setting to 0.0.'
                rms(ichan,iepoch) = 0.
            endif
c
c Excessive RMS detection:
c
            if (rms(ichan,iepoch).gt.rmscrit) then
                rmsart(ichan,iepoch) = 1
            endif
        enddo
c
c Dead channel detection:  Note that eye channels are never declared dead.
c
        if ((ichan.ne.LEYECHAN).and.(ichan.ne.VEYECHAN)) then
            do iepoch = firstepoch, lastepoch
                if (rms(ichan,iepoch).lt.deadcrit) then
                        deadchan(ichan,iepoch) = 1
                endif
            enddo
```

```
        endif
c
c END PRE-CORRECTION ARTIFACT CHECK
c
c*******************************************************************************
c
c BEGIN EYE MOVEMENT FILTER AND FINAL ARTIFACT CHECK.
c
c The eeg is checked for absolute amplitude artifacts at the end of this
c block of code, after subtracting the scaled lateral eog from the eeg
c
c Blink correction:
c
c Regress the eeg on the eog in regions where the eog is changing rapidly
c and get the proportionality constants, b(c), over all channels.  Epochs
c with spikes in the eeg or clipping in the eog or eeg are not used.
c
c An inefficiency of this section of code is that blinks are detected each
c time the code is executed.
c
c The lateral eog is corrected first.
c
        if (ichan.ne.VEYECHAN) then
            if (verbose) then
                write (STDOUT,*) '    Regressing EEG on EOG...'
            endif
c
c Process each epoch separately
c
            iepoch = firstepoch - 1
            do nominale = nominalfirst, nominallast
                iepoch = iepoch + 1
                if ( (blink(VEYECHAN,iepoch)   .eq. 1) .and.
     .               (spike(ichan,iepoch)      .eq. 0) .and.
     .               (clipart(VEYECHAN,iepoch) .eq. 0) .and.
     .               (clipart(ichan,iepoch)    .eq. 0)) then
c
c Place the pointers at the proper place in the vector
c
                    firstpoint = (nominale - 1) * np + 1
                    lastpoint = (firstpoint + np) - 1
c
c Form the regression sums for this epoch
c
                    cov = 0.
                    ss = 0.
                    tx = 0.
                    ty = 0.
                    nn = 0
                    do ipoint = firstpoint+delta, lastpoint-delta
                        if (abs(v2(ipoint+delta)-v2(ipoint-delta)).gt.
     .                      blinkcrit) then
c
                            v1adj = v1(ipoint)
                            v2adj = v2(ipoint)
```

```
c
                                tx = tx + v2adj
                                ty = ty + v1adj
                                nn = nn + 1
                                cov = cov + v1adj * v2adj
                                ss = ss + v2adj * v2adj
                            endif
                        enddo
c
c Calculate the blink scaling constants for each channel
c
                        b(ichan) = 0.0
                        if (ss.eq.0.) then
                            if (nn.gt.0) then
                                write (STDERR,*)
     .                          'artfil: Attempt to divide VRTEYE ',
     .                          'covariance by zero at channel ',ichan
                            endif
                        else
                            if (nn.gt.0) then
                                ss = ss - (tx**2.)/float(nn)
                                cov = cov - (tx*ty)/float(nn)
                                if (ss.ne.0.0) then
                                    b(ichan) = cov / ss
                                endif
                            endif
                        endif
c
c Apply the correction factor.  If something went wrong with the calculation
c of b(), then it was set to zero, causing no change to the data.
c
                        do ipoint = firstpoint, lastpoint
                            v1(ipoint) = v1(ipoint) - b(ichan)*v2(ipoint)
                        enddo
                    endif   ! blink
                enddo       ! nominale
            endif           ! .not. VEYECHAN
c
c End blink correction
c
c If the Leog was just blink corrected, replace the uncorrected data
c in V3 with corrected data.
c
            if (ichan .eq. LEYECHAN) then
                do i = 1, nptot
                    v3(i) = v1(i)
                enddo
            endif
c
c Lateral eye movement correction:
c
c The lateral eye channel is corrected for blinks on the first pass.  The
c blink corrected lateral eog channel is used to correct the eeg
c thereafter.  Once again, epochs with spikes or clipping are not used.
c Also, once again, the lateral eog is not corrected for lateral eye movement.
```

```
c
        if (ichan.ne.LEYECHAN) then
            cov = 0.
            ss = 0.
            tx = 0.
            ty = 0.
            nn = 0
            iepoch = firstepoch - 1
            do nominale = nominalfirst, nominallast
                iepoch = iepoch + 1
                if ((spike(LEYECHAN,iepoch)    .eq. 0) .and.
     .              (spike(ichan,iepoch)       .eq. 0) .and.
     .              (clipart(LEYECHAN,iepoch) .eq. 0) .and.
     .              (clipart(ichan,iepoch)     .eq. 0)) then
                    firstpoint = (nominale - 1) * np + 1
                    lastpoint = firstpoint + np - 1
                    do ipoint=firstpoint+delta, lastpoint-delta
                        tx = tx + v3(ipoint)
                        ty = ty + v1(ipoint)
                        nn = nn + 1
                        cov = cov + v1(ipoint) * v3(ipoint)
                        ss = ss + v3(ipoint) * v3(ipoint)
                    enddo
                endif
            enddo
c
            if (ss.eq.0.) then
                if (nn.gt.0) then
                    write (STDERR,*)  'artfil: Attempt to divide LATEYE ',
     .                  'covariance by zero at channel ', ichan
                endif
                b(ichan) = 0.0
            else
                if (nn.gt.0) then
                    ss = ss - (tx**2.)/float(nn)
                    cov = cov - (tx*ty)/float(nn)
                    b(ichan) = cov / ss
                else
                    b(ichan) = 0.
                endif
            endif
c
c Subtract the scaled lateral eog and check the resulting data
c for absolute amplitude artifacts.  Epochs with spikes or clipping
c are not corrected.  Also, the "delta" points at the beginning
c and end of each vector are not checked for artifacts.  The lateral
c eog is not checked for amplitude artifacts.
c
            if (abs(b(ichan)) .gt. 0.) then
                iepoch = firstepoch - 1
                do nominale = nominalfirst, nominallast
                    iepoch = iepoch + 1
c
c Check for identified artifacts:
c
```

```
                    if ((spike(LEYECHAN,iepoch)   .eq. 0) .and.
     .                  (spike(ichan,iepoch)      .eq. 0) .and.
     .                  (clipart(LEYECHAN,iepoch) .eq. 0) .and.
     .                  (clipart(ichan,iepoch)    .eq. 0)) then
c
c Subtract the scaled eog if the epoch is good:
c
                        firstpoint = (nominale-1)*np+1
                        lastpoint = firstpoint+np-1
                        do ipoint = firstpoint, lastpoint
                            v1(ipoint) = v1(ipoint)-b(ichan)*v3(ipoint)
                        enddo
c
c Check the corrected eeg for large absolute voltages:
c
                        if (ichan .ne. LEYECHAN) then
                            do ipoint = firstpoint+delta, lastpoint-delta
                                if (abs(v1(ipoint)).gt.voltcrit) then
                                    voltart(ichan,iepoch) = 1
                                endif
                            enddo
                        endif   ! .not. lateral eye channel
                    endif       ! no spike artifacts
                enddo           ! epoch
            endif
        endif
c
c END LATERAL EOG CORRECTION
c
c END EYE MOVEMENT FILTER
c
c Convert to integer*2, rescale (converts to uV), and write
c corrected EEG to the output file
c
        if (verbose) then
            print *,'Writing all epochs for channel ',ichan
        endif
        ptemp = 0
        do iepoch = firstepoch, lastepoch
            do ipoint = 1, np
                ptemp = ptemp + 1
c
c The following results are rounded before truncating
c
                if (v1(ptemp) .ge. 0) then
                    dummy(ipoint) = v1(ptemp) * 10. + .5
                else
                    dummy(ipoint) = v1(ptemp) * 10. - .5
                endif
            enddo
c
            recno = nctot * (iepoch-1) + ichan
            call quickio (OUTFILE, outpath(1:lenoutpath), 'unk',
     .          'write', recno, dummy, np, ios)
            if (ios.ne.0) then
```

```
                write (STDERR,*) 'artfil: Error writing to output ',
     .                           'file, iostat is ',ios
                goto 32767
            endif
        enddo
c
c End of channel loop
c
        goto 150        ! process next channel
c
c Finished with channels, clean up
c
3100    continue
c
c Fill out the final records for the last epoch
c
        do ipoint = 1, np
            dummy(ipoint) = 0
        enddo
c
        recno = nctot * lastepoch
        call quickio (OUTFILE, outpath(1:lenoutpath), 'unk', 'write', recno,
     .      dummy, np, ios)
        if (ios.ne.0) then
            write (STDERR,*) 'artfil: Error writing cleanup records, ',
     .                       'iostat is ',ios
            goto 32767
        endif
c
c Close the data files
c
        close (INFILE)
        close (OUTFILE)
c
c Consolidate some of the artifacts into a summary variable to be used
c for deleting epochs in subsequent processing.  Channels within epochs
c are deletable if they contain clipping, large absolute or rms voltages
c in any channel except the vertical eog channel.  If there is clipping
c in the vertical or lateral eog, the entire epoch is marked for deletion.
c
        if (verbose) then
            print *,'Creating artifact summary file'
        endif
        do ichan = 1, nc
            do iepoch = firstepoch, lastepoch
                if ((ichan.eq.LEYECHAN).and.(clipart(ichan,iepoch).eq.1)) then
                    if (verbose) then
                        print *,'Epoch ',iepoch,' Chan. ',ichan,
     .                      ' - Lateral eye channel clipping detected'
                    endif
                    sumart(ichan,iepoch) = 1
                endif
c
                if ((ichan.eq.VEYECHAN).and.(clipart(ichan,iepoch).eq.1)) then
                    if (verbose) then
```

```
                print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Vertical eye channel clipping detected'
              endif
              sumart(ichan,iepoch) = 1
            endif
c
            if ((ichan.eq.LEYECHAN).and.(spike(ichan,iepoch).eq.1)) then
              if (verbose) then
                print *,'Epoch ',iepoch,' Chan. ',ichan,
     .          ' - Lateral eye channel spike detected'
              endif
              sumart(ichan,iepoch) = 1
            endif
c
            if ((ichan.ne.LEYECHAN).and.(ichan.ne.VEYECHAN))then
              if (spike(ichan,iepoch).eq.1) then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Spike detected'
                endif
                sumart(ichan,iepoch) = 1
              endif
              if (voltart(ichan,iepoch).eq.1) then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Voltage artifact detected'
                endif
                sumart(ichan,iepoch) = 1
              endif
              if (clipart(ichan,iepoch).eq.1) then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Clipping detected'
                endif
                sumart(ichan,iepoch) = 1
              endif
              if (rmsart(ichan,iepoch).eq.1) then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Excessive RMS detected'
                endif
                sumart(ichan,iepoch) = 1
              endif
              if (clipart(LEYECHAN,iepoch).eq.1) then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Artifact due to lateral eye clipping'
                endif
                sumart(ichan,iepoch) = 1
              endif
              if (clipart(VEYECHAN,iepoch).eq.1)then
                if (verbose) then
                  print *,'Epoch ',iepoch,' Chan. ',ichan,
     .              ' - Artifact due to vertical ',
     .              'eye clipping'
```

```
                    endif
                    sumart(ichan,iepoch) = 1
                endif
            endif
        enddo
    enddo
c
c Create the artifact summary file
c
    artpath = 'artifacts'
    call removespaces (artpath,lenartpath)
c
    open (ARTFILE, file=artpath(1:lenartpath), status='unk',
   .    access='dir', recl=20, iostat=ios)
    if (ios.ne.0) then
        write (STDERR,*) 'artfil: Error opening artifact file, ',
   .                     'iostat is ',ios
        goto 32767
    endif
c
    do iepoch = firstepoch, lastepoch
        do ichan = 1, nc
            arts(1) = sumart(ichan,iepoch)
            arts(2) = iepoch
            arts(3) = ichan
            arts(4) = blink(ichan,iepoch)
            arts(5) = voltart(ichan,iepoch)
            arts(6) = rmsart(ichan,iepoch)
            arts(7) = deadchan(ichan,iepoch)
            arts(8) = spike(ichan,iepoch)
            arts(9) = clipart(ichan,iepoch)
            arts(10) = rms(ichan,iepoch) + .5
            recno = (iepoch - 1) * nctot + ichan
            write (ARTFILE, rec=recno, iostat=ios) arts
            if (ios.ne.0) then
                write (STDERR,*) 'artfil: Error writing to ',
   .                             ' artifact summary file, ',
   .                             ' iostat is ',ios
                goto 32767
            endif
c
c Fill out the records for the last epoch, if neccessary.
c
            if ( (iepoch.eq.lastepoch).and.
   .            (ichan.eq.nc).and.(nc.ne.nctot)) then
                do i = 1, 10
                    arts(i) = 0
                enddo
                recno = nctot * lastepoch
                write (ARTFILE, rec=recno, iostat=ios) arts
                if (ios.ne.0) then
                    write (STDERR,*) 'artfil: Error writing ',
   .                                 'cleanup records to artifact ',
   .                                 'summary file, iostat is ',ios
                    goto 32767
```

```
                    endif
                endif
            enddo
        enddo
        close (ARTFILE)
32767   stop
        end

*******************************************************************************

        subroutine smooth3p (dummy, rdummy, np)
c
c Converts integer*2 vectors to real, divides each element by 10,
c and computes a three-point smooth.
c
        integer*2 dummy(np)
        real rdummy(np)
        integer np, p
        real z1, z2, z3, k1, k2
c
c The smoothing algorithm is x(t) = .25x(t-2)+.5x(t)+.25x(t+1)
c
        k1 = .025
        k2 = .05
        z2 = float(dummy(1))
        z3 = float(dummy(2))
c
        do p = 2, np - 1
            z1 = z2
            z2 = z3
            z3 = float(dummy(p+1))
            rdummy(p) = k1 * z1 + k2 * z2 + k1 * z3
        enddo
c
c Set the first and last elements of the smoothed vector
c the values of their immediate neighbors.
c
        rdummy(1) = rdummy(2)
        rdummy(np) = rdummy(np-1)
c
        return
        end

*******************************************************************************

        subroutine transparm (parmstr,retoption,rettype,
     .          retival,retrval,retcval)
        integer nnumind,nintind
        parameter       (nnumind = 15)
        parameter       (nintind = 12)
c
        character*(*) parmstr
        character*256 retcval,parm
        character*1 rettype,retoption,
     .          numind(nnumind),intind(nintind)
```

```
c
        integer retival,entirelength,parmlength
        integer i,j,ios
        real retrval
c
        data numind /'0','1','2','3','4','5','6','7','8','9','E','e',
                    '-','+','.'/
        data intind /'0','1','2','3','4','5','6','7','8','9','+','-'/
c
c Given a parameter string in the format '-' option '...string...'
c
c       a) extract the option (one character).
c       b) determine the type of the string as integer, real, or character.
c       c) convert integer and real values to internal representation.
c       d) return values to calling program.
c
c Author:  Sam J. La Cour, Jr.
c Date:    February 20, 1987
c
c First, get the length of the entire string.  In order for it to be a valid
c option string, the length must be at least 2.
c
        call getlen (parmstr,i)
        if (i.lt.2) then
            retoption = ' '
            rettype = 's'
            return
        endif
c
c Next, see if it is a valid option parameter string.  In order for this to
c be true, the first character must be a '-', followed by an option character,
c followed by an optional string.
c
        if (parmstr(1:1).ne.'-') then
            retoption = ' '
            rettype = 'u'
            return
        endif
c
c Ok, we have a '-', let's get the option character.  It can be anything
c except a blank.
c
        retoption = parmstr(2:2)
c
c Now that we have a legal option character, the remainder of the option
c string needs to be parsed, if present.  If the length of the option
c string is equal 2, just go ahead and exit.
c
        if (entirelength.eq.2) then
            rettype = 'n'
            return
        endif
c
c Get the actual length of the rest of the parameter string.
c
```

```
        parm = parmstr(3:)
        call getlen (parm,parmlength)
        if (parmlength.le.0) then
            rettype = 'n'
            return
        endif
c
c Scan the parameter string for non-numeric characters.
c For our purposes, numeric characters are '0'..'9', 'e','E','-','+' and '.'
c All other characters classify the string as non-numeric, and therefore
c the type is forced to be character.  After this is done, if the the type is
c determined to be possibly numeric, a further scan is done to eliminate the
c 'e','E' and '.'.  This filter will determine if the number is possibly an
c integer value.  Once this is done, internal reads will be used to do the
c actual conversion from string to internal format.  Any failures in the
c conversion, due to things like '1.0ee1.3-', will be flagged here.  If all
c is successful, the type and value will have been determined and
c returned to the main program.
c
c Pass through any numerics, send all others to type CHARACTER
c
        do 1010 i=1,parmlength
        do 1020 j=1,nnumind
        if (parm(i:i).eq.numind(j)) goto 1010
1020    continue
c
c A match with the numeric values was not found, the string must
c be of type CHARACTER, therefore, proceed to that section.
c
        goto 4000
1010    continue
c
c All characters in the string passed the numeric test, now lets
c try to find out if the number is integer or real by looking for
c characters which would indicate that the value was a real number,
c such as decimal points or exponentials.
c
c Pass through any integers, send all others to type REAL
c
        do 2010 i=1,parmlength
        do 2020 j=1,nintind
        if (parm(i:i).eq.intind(j)) goto 2010
2020    continue
c
c A match with the integer values was not found, the string must
c be of type REAL, therefore proceed to that section.
c
        goto 3000
2010    continue
c
c The value has only legal integer characters in it, namely '0'..'9' and '-','+'.
c Try to convert it.  Any errors are due to incorrect format.
c
        read (parm(1:parmlength),*,iostat=ios) retival
        if (ios.ne.0) then
```

```
            rettype = 'u'
        else
            rettype = 'i'
        endif
        return
c
c The value has only legal real characters in it, namely '0'..'9','e','E','+',
c '-','.'.  Try to convert it.
c
3000    continue
        read (parm(1:parmlength),*,iostat=ios) retrval
        if (ios.ne.0) then
            rettype = 'u'
        else
            rettype = 'r'
        endif
        return
c
c Character is the only type left (a trash disposal...)
c
4000    continue
        rettype = 'c'
        retcval = parm
        return
        end

*******************************************************************************

        subroutine removespaces (instring, newlen)
c
c Removes spaces and nulls from a string and determines the length of the string
c
        character*(*) instring
        character*256 temp
        integer i, newlen
c
        newlen = 0
        temp = ' '
        do 1000 i=1,len(instring)
        if (instring(i:i).ne.' '.and.instring(i:i).ne.char(0)) then
            newlen = newlen + 1
            temp(newlen:newlen) = instring(i:i)
        endif
1000    continue
        instring = temp
        return
        end

*******************************************************************************

        subroutine getlen (str,strlength)
c
c Given any length string, returns the length of the string with all
c trailing spaces removed.  In other words, the length of the string
c scanned from the right until the first non blank character.
```

```
c
c Examples:
c
c       'foo bar' returns a length of 7
c       'foo bar ' also returns a length of 7
c       '  foo bar' returns a length of 9
c
c Author:       Sam J. La Cour, Jr.
c Date:         February 20, 1987
c
        character*(*) str
        integer strlength,i,j
c
c Ignore the trivial case and exit
c
        j = len(str)
        if (j.le.0) then
            strlength = 0
            return
        endif
c
c Scan from the right, constantly updating 'strlength'. Exit when the
c entire string has been scanned.  If no non-blanks are found, the length
c will be zero.
c
        strlength = j
        do 1000 i=1,j
            if (str(strlength:strlength).ne.' ') go to 32767
            strlength = strlength - 1
1000    continue
32767   return
        end

*******************************************************************************

        subroutine casefold (string)
c
c Routine to change all characters in a string to upper case for comparison
c This is most useful on things like "y" or "Y" inputs, where it doesn't matter
c what the case of the response is.
c
        character*(*) string
        integer i,j,l
c
        l = len(string)
        do 1000 i=1,l
        j = ichar(string(i:i))
        if (j ge.97.and.j.le.122) then
            j = j - 32
            string(i:i) = char(j)
        endif
1000    continue
        return
        end
```

```
*************************************************************************

        subroutine quickio (unit,file,status,operation,
     .          recno,buffer,size,ios)
c
        character*(*) file,status,operation
        character*20 tempop
        integer unit,recl,ios,recno,size
        integer*2 buffer(size)
c
        tempop = operation
        call casefold (tempop)
        if (tempop.eq.'OPEN') then
            if (recl.gt.0) then
                recl = size * 2
                open (unit=unit,file=file,status=status,access='direct',
     .                  recl=recl,iostat=ios)
            endif
        else if (tempop.eq.'READ') then
            read (unit=unit,rec=recno,iostat=ios) buffer
        else if (tempop.eq.'WRITE') then
            write (unit=unit,rec=recno,iostat=ios) buffer
        endif
        return
        end
```

Other Related NAMRL Publications

Reeves, D.L. and Gadolin, R.E., Sustained/Continuous Operations Subgroup of the Department of Defense Human Factors Engineering Technical Group: Program Summary and Abstracts from the 8th Semiannual Meeting, NAMRL Technical Memorandum 89-2, Naval Aerospace Medical Research Laboratory, Pensacola, FL, April 1989.